J Grid Computing (2012) 10:367–393
DOI 10.1007/s10723-012-9222-7

A Distributed Workflow Management System with Case Study of Real-life Scientific Applications on Grids

Qishi Wu · Mengxia Zhu · Yi Gu · Patrick Brown · Xukang Lu · Wuyin Lin · Yangang Liu

Received: 30 September 2011 / Accepted: 6 July 2012 / Published online: 16 August 2012

Abstract Next-generation scientific applications feature complex workflows comprised of many computing modules with intricate inter-module dependencies. Supporting such scientific workflows in wide-area networks especially Grids and optimizing their performance are crucial to the success of collaborative scientific discovery. We develop a Scientific Workflow Automation and Management Platform (SWAMP), which enables scientists to conveniently assemble, execute, monitor, control, and steer computing workflows in distributed environments via a unified web-based user interface. The SWAMP architecture is built entirely on a seamless composition of web services: the functionalities of its own are provided and its interactions with other tools or systems are enabled through web services for easy access over standard Internet protocols while being independent of different platforms and programming languages. SWAMP also incorporates a class of efficient workflow mapping schemes to achieve optimal end-to-end performance based on rigorous performance modeling and algorithm design. The performance superiority of SWAMP over existing workflow mapping schemes is justified by extensive simulations, and the system efficacy is illustrated by large-scale experiments on real-life scientific workflows for climate modeling through effective system implementation, deployment, and testing on the Open Science Grid.

Keywords Distributed computing · Scientific workflow · Climate modeling · Open Science Grid

Q. Wu (✉) · X. Lu
Department of Computer Science,
University of Memphis,
Memphis, TN 38152, USA
e-mail: qishiwu@memphis.edu

X. Lu
e-mail: xlv@memphis.edu

M. Zhu · P. Brown
Department of Computer Science, Southern Illinois University, Carbondale, IL 62901, USA

M. Zhu
e-mail: mzhu@cs.siu.edu

P. Brown
e-mail: patiek@cs.siu.edu

Y. Gu
Dept of Management, Marketing,
Computer Science, and Information Systems,
University of Tennessee at Martin,
Martin, TN 38238, USA
e-mail: ygu6@utm.edu

W. Lin · Y. Liu
Atmospheric Science Division, Brookhaven National Laboratory, Upton, NY 11793, USA

W. Lin
e-mail: wlin@bnl.gov

Y. Liu
e-mail: lyg@bnl.gov

1 Introduction

Next-generation e-science features complex workflows comprised of many computing modules (also referred to as activities, stages, jobs, transformations, or subtasks in different contexts) with intricate inter-module execution dependencies. The execution of such scientific workflows typically involves the invocation of a large number and wide variety of distributed programs or tools for collaborative data analysis and knowledge discovery [60], and also requires the use of a wide range of expensive and powerful resources including supercomputers, PC clusters, high-end workstations, experimental facilities, large-screen display devices, high-performance network infrastructures, and massive storage systems. Typically, these resources are deployed at various research institutes and national laboratories, and are provided to users through wide-area network connections that may span through several countries [10, 19, 36, 61], hence inevitably exhibiting an inherent dynamic nature in their availability, reliability, capacity, and stability.

As new computing technologies and networking services rapidly emerge, enabling functionalities are progressing at an ever-increasing pace, unfortunately, so are the dynamics, scale, heterogeneity, and complexity of the networked computing environment. Supporting large-scale scientific workflows in distributed heterogeneous network environments is crucial to ensuring the success of mission-critical scientific applications and maximizing the utilization of massively distributed computing and networking resources. However, application users, who are primarily domain experts, oftentimes need to manually exploit available resources, and configure and run their computing tasks over networks using software tools they are familiar with based on their own empirical studies, inevitably leading to unsatisfactory performance in such diverse and dynamic environments.

We propose to develop a Scientific Workflow Automation and Management Platform (SWAMP[1]) to support the distributed execution and management of large-scale scientific workflows in heterogeneous network environments. Compared to the previous work in [76], we have made significant and substantial changes to both the design and implementation of the system:

- The entire SWAMP architecture has been completely redesigned and re-implemented using distributed web services for a seamless integration instead of individual software packages installed on the user site.
- The workflow execution mode has been changed from Condor's default centralized data forwarding mechanism to the distributed direct inter-module data transfer using Stork for improved end-to-end performance.
- The previous system employs an extremely simplified modeling method that uses the processor frequency to represent the node's processing power; while the new system takes a rigorous statistical approach that considers a combination of both hardware and software properties to model and predict the performance of various types of scientific computations with a higher accuracy.
- In addition to the algorithm for delay minimization in unitary processing applications, the workflow mapping component in the new system has incorporated a set of specially-designed algorithms for throughput maximization in streaming applications.

Within SWAMP, a scientific workflow is modeled as a Directed Acyclic Graph (DAG) where each vertex represents a computing module and each directed edge represents the execution dependency between two adjacent modules. Each module is considered as an autonomous

[1] A preliminary version of the SWAMP development [76] was published in Proc. of the 6th Int. Workshop on Sys. Man. Tech., Proc., and Serv., in conjunction with IPDPS, Atlanta, GA, April 19, 2010.

computing agent: it receives data as input from one or more preceding modules, executes a predefined processing routine on the aggregate input data, and sends the results as output to one or more succeeding modules. We assume that neither can a module start execution until all the required inputs arrive nor send out the results until the execution is completed on each dataset. The current version of SWAMP integrates a graphical toolkit based on a unified web interface for abstract workflow composition, employs Condor/DAGMan [12] for workflow dispatch, and adopts Stork [64] for direct inter-module data transfer to realize completely distributed execution without centralized data forwarding. The goal of SWAMP is to pool geographically distributed but Internet connected computing resources together to enable and accelerate the execution of computation-intensive scientific workflows in wide-area networks.

The SWAMP architecture is entirely built on a seamless composition of web services: the functionalities of its own are provided and its interactions with other tools or systems are facilitated through web services for easy access over standard Internet protocols while being independent of different platforms and programming languages. SWAMP also incorporates a specially-designed mapping engine that automatically maps abstract workflows to underlying networks to achieve *Minimum End-to-end Delay* (MED) for unitary (one-time) processing applications and *Maximum Frame Rate* (MFR) for streaming applications based on real-time network measurements and system status monitoring [28, 73]. The core of this mapping engine consists of a class of workflow mapping optimization schemes developed through rigorous performance modeling and algorithm design. The performance superiority of SWAMP over existing workflow mapping schemes is justified by extensive simulations, and the system efficacy is illustrated by large-scale workflow experiments on a real-life scientific application for climate modeling through effective system implementation, deployment, and testing on the Open Science Grid [48].

The rest of the paper is organized as follows: In Section 2, we conduct a survey of related work. The SWAMP framework and functional components are described in Section 3. The performance modeling and prediction approach is introduced in Section 4. The workflow mapping algorithms are presented in Section 5. The system implementation details and performance evaluations are provided in Section 6. We conclude our work in Section 7.

2 Related Work

With the advent of next-generation e-science, a plethora of frameworks and tools have been developed for generating, refining, and executing scientific workflows. Such efforts include P-GRADE [33], Pegasus [15], Kepler [40], Condor/DAGMan [12], Taverna [30], Triana [9], Swift [65], Java CoG Kit [37], Sedna [71], and SWAMP [76]. Each workflow system has its own language, design suite, and software components, and the systems vary widely in their execution models and the kinds of components they coordinate [14]. Some systems attempt to provide general-purpose workflow functionalities while others are more geared toward specific applications and are optimized to support specific component libraries. Google's MapReduce provides a software framework to support distributed computing on clusters of computers [13]. Since its primary goal is to process large datasets through a partition-composition approach using the map and reduction operations, MapReduce does not sufficiently meet the requirement of distributed scientific workflows that typically consist of different computational jobs executed in wide-area networks.

Many existing systems support workflow execution in Grid environments, which provide compositional programming and execution models to enable resource interactions by supplying the mechanisms to access, aggregate, and manage the network-based infrastructure of science [32]. Such Grid-based scientific applications include Large Hadron Collider (LHC) supported by the

Worldwide LHC Computing Grid (WLCG) [72], climate modeling supported by the Earth System Grid (ESG) [20], and NASA's National Airspace Simulation supported by the Virtual National Airspace Simulation (VNAS) Grid [32, 43]. Many other Grid systems such as Open Science Grid (OSG) [48], ASKALON [21, 66], TeraGrid, and GridCOMP (see [17] for a more complete list) are under rapid development and deployment. These existing workflow or Grid systems have a primary design goal to provide services or infrastructures for coordinated application interoperation, distributed job submission, or large data transfer, but generally there lacks a comprehensive workflow solution that integrates workflow mapping schemes to support large-scale scientific applications for end-to-end performance optimization. Furthermore, most systems using a batch scheduler are not inherently capable of supporting streaming applications that require computational steering, which is a critical activity in explorative sciences where the parameter values of an online simulation or a computing module must be changed and determined on a realistic time scale.

Over the years, the workflow mapping problems in heterogeneous network environments have been increasingly identified and investigated by many researchers to facilitate more convenient and comprehensive collaborations among different institutes and domains across wide-area networks [6, 8, 16, 31, 55]. Existing workflow mapping algorithms can be roughly classified into the following five categories [42, 69]: (i) Graph-based methods [11, 44]. Among the traditional graph mapping problems in theoretical aspects of computing, subgraph isomorphism is known to be NP-complete while the complexity of graph isomorphism still remains open. (ii) List scheduling techniques, in which the most commonly used is critical path method [34, 41, 54]. (iii) Clustering algorithms [6, 24], which assume an unlimited number of processors and thus are not very feasible for practical use. (iv) Duplication based algorithms [3, 7, 55], most of which are of a prohibitively high complexity. (v) Guided random search methods such as genetic algorithm [4, 70] and simulated annealing [59], where additional efforts are often required to determine an appropriate termination condition and usually there is no performance guarantee.

Most workflow mapping or task scheduling problems in Grid environments or multiprocessor systems assume complete networks, homogeneous resources, and specific mapping constraints. Our work differs from the above efforts in several aspects: (i) We investigate the workflow mapping problems for both MED and MFR. (ii) We target computer networks of arbitrary topology with heterogeneous computer nodes and network links. (iii) We consider resource sharing among multiple concurrent module executions on the node or data transfers over the link. (iv) We employ a statistical approach to performance modeling and prediction. (v) We build the SWAMP system on web services for easy access and enhanced interoperability across different platforms. (vi) We implement and integrate our mapping optimization algorithms into the SWAMP system and test them with real-life scientific workflows on Grids.

3 SWAMP System Design

SWAMP enables application users to conveniently assemble, execute, monitor, and control complex computing workflows in heterogeneous high-performance network environments. As shown in Fig. 1, SWAMP provides these services through the interactions between three main elements: (a) Kepler Manager, (b) Web Server Manager, and (c) DAGMan Manager.

Within SWAMP, a user can use either a graphical web interface or the GUI of Kepler to compose abstract workflows. The Kepler Manager converts the abstract workflow in XML format to DAG format, and sends these workflow description files including a meta-workflow and a list of component workflows to the Web Server Manager. A meta-workflow file may contain one component workflow with a single dataset or more with multiple (e.g. time-series) input datasets. The Web Server provides a visual management interface for workflow selection, dispatch, monitoring, and steering. The user can select a subset of

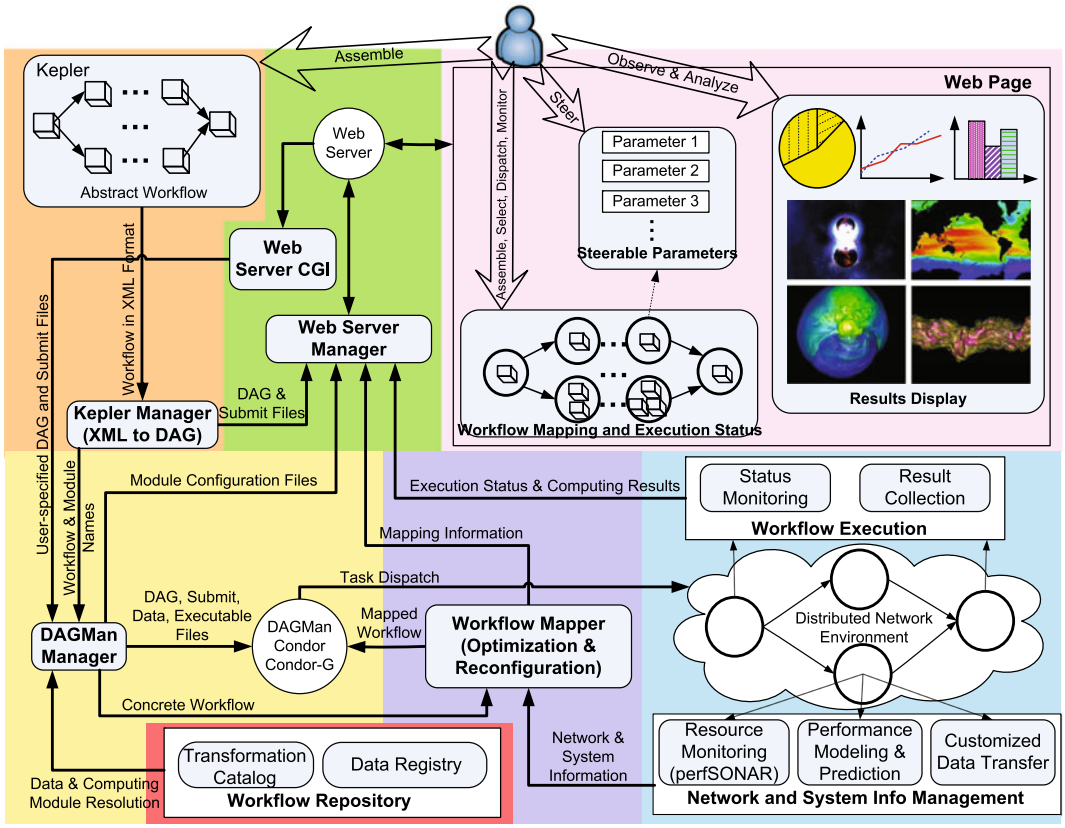

Fig. 1 The SWAMP framework: a high-level overview

component workflows of interest, configure their parameter settings, and dispatch them for execution through a web browser.

Upon the receival of an abstract workflow from the Web Server, the DAGMan Manager invokes the Workflow Mapper to map the abstract workflow to the real network based on the availability and capability of computer nodes collected by the Network and System Information Management component. The mapped abstract workflow is then submitted to Condor or Condor-G, which extracts executable modules and source datasets from the workflow repository and composes them into an executable workflow. The mapped executable workflow is dispatched to the network for execution. Note that Condor/DAGMan has a default centralized execution model where the output of each module is sent back to the submitter for forwarding to its succeeding module. This centralized model may lead to prohibitively heavy traffic in the network, especially for data-intensive workflows with many computing modules. We realize a completely distributed execution model by adopting Stork [64] for direct inter-module data transfer. The workflow execution status information as well as the final results are collected and sent on the fly to the Web Server Manager for display on the web page. Based on the displayed results, the user may reset the value of a command argument for those steerable computing modules and re-dispatch them into the network.

Figure 2 shows the SWAMP architecture, where the main function components are implemented as web services to achieve a seamless integration, and the solid and dashed arrows denote data flows and control flows, respectively. More details of the SWAMP design and implementation are provided below.

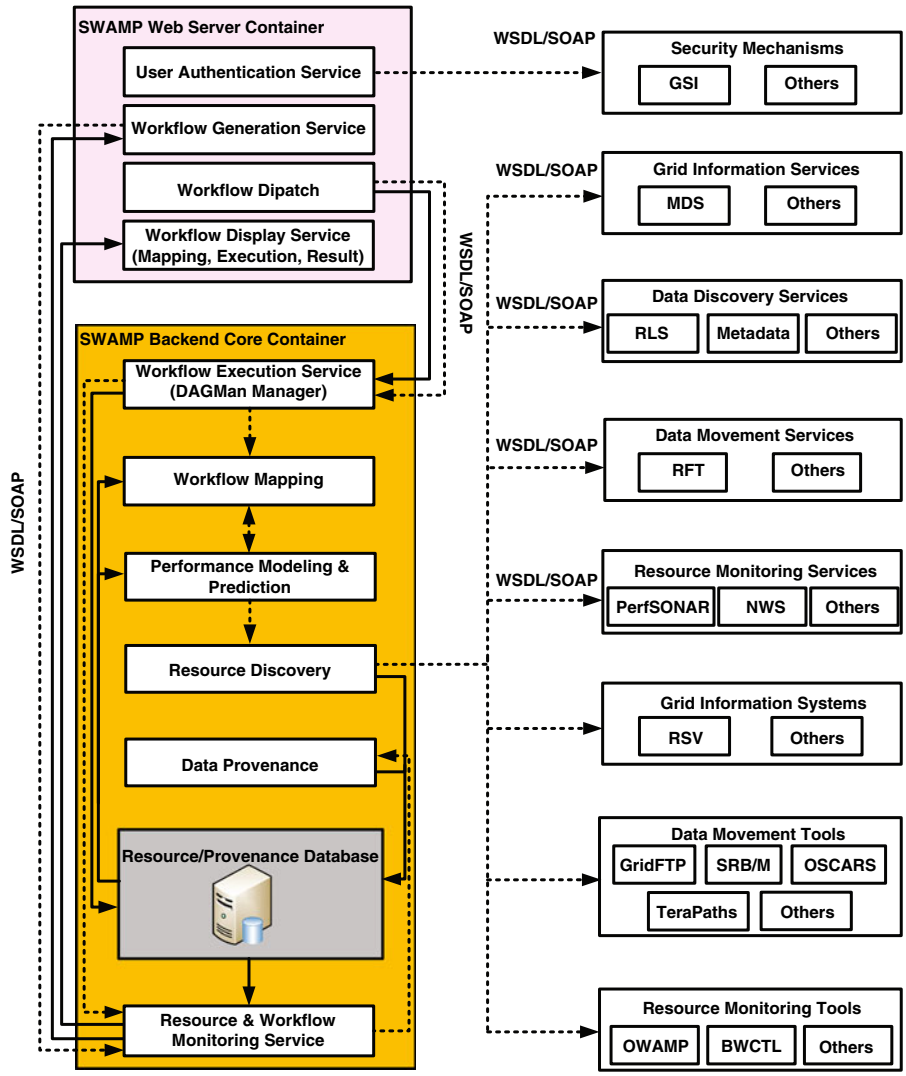

Fig. 2 The SWAMP architecture: web services-based function components, control and data flows

3.1 User Authentication Service

Different computing environments may have distinct usage policies including authentication and authorization. The user authentication component in SWAMP is designed as a web service to support a wide variety of security mechanisms such as Grid security infrastructure (GSI), Kerboros, simple login/password, SSL, and so on.

3.2 Workflow Generation Service

In addition to the GUI of Kepler, SWAMP provides a graphical toolkit for visual construction of abstract workflows on a unified web interface, which generates a DAG file and a set of corresponding submit files required by Condor DAGMan. As illustrated in Fig. 3, the user can interactively compose complex abstract workflows by simply dragging, dropping, and connecting modules and specifying their detailed configurations. This web interface also supports the creation of a meta-workflow in XML for easy human interpretation and machine parsing. A meta-workflow describes a list of component workflows that can be submitted as a full set, subset, or individually depending on the needs of the user.

However, it may not be always convenient or feasible for the user to construct workflows via

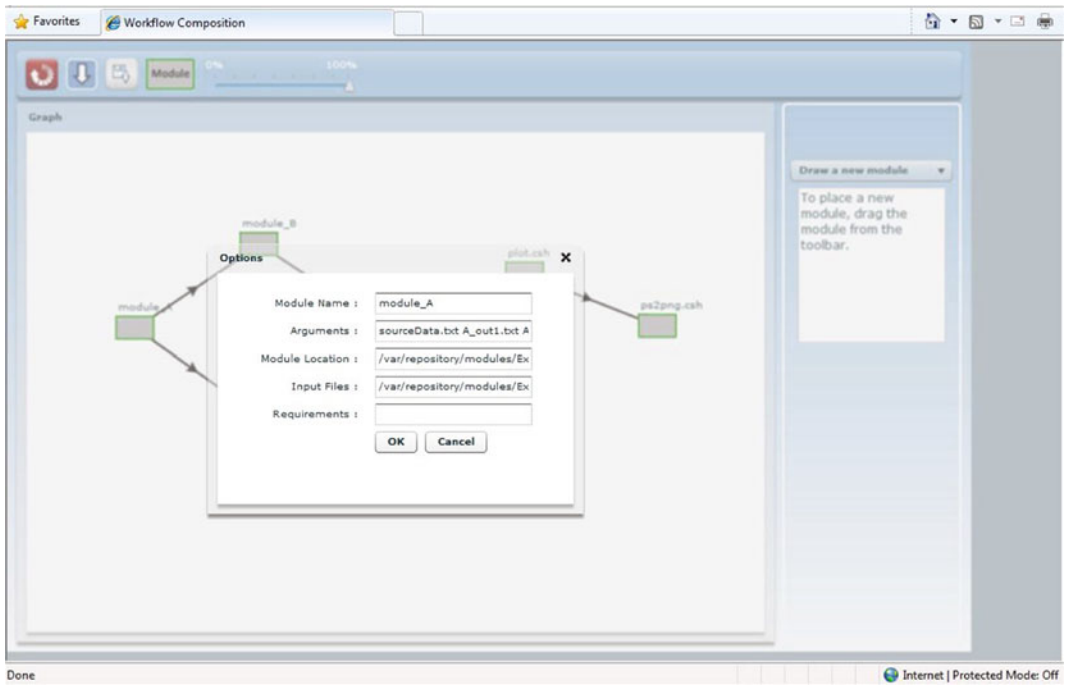

Fig. 3 The web-based workflow generation by dragging, dropping and connecting modules and specifying their detailed configurations

the graphical toolkit in the web browser for applications where a large number of modules are involved, or a collection of workflows need to be generated by combining the parameters in a particular way specified by domain experts. SWAMP provides a workflow generation web service to automatically generate such workflows. The inputs to this web service are partial workflow representations that describe the input files, the executables (workflow components) and their parameters, as well as the output files produced by these executables. For example, the workflow structure and module information specified through the web interface in Fig. 3 can be directly sent to the web service for automatic workflow generation. This service also requests the information on datasets from the web interface and available computing resources from the resource and workflow monitoring service.

The outputs of both interactive and automatic workflow generations are workflow description files in XML format. Scientists are also allowed to customize the workflows by modifying the steerable parameters through the web interface after the workflows are dispatched.

3.3 Workflow Execution Service

The workflow execution component is implemented as a web service, which receives the abstract workflow description files from its client, i.e. the workflow dispatch component. The workflow execution service needs to contact resource and workflow monitoring service to place callback to the workflow display service for dispatched workflows. If the workflow execution service fails to place the callback, it will prevent the workflow from dispatching. Otherwise, the workflow execution service proceeds to perform workflow mapping, generates the desired workflow and job description files, and dispatches the workflows to the Condor environment. The workflow execution service serves as a portal to the Condor batch system and its main goal is to simplify the process of real workflow submission and execution.

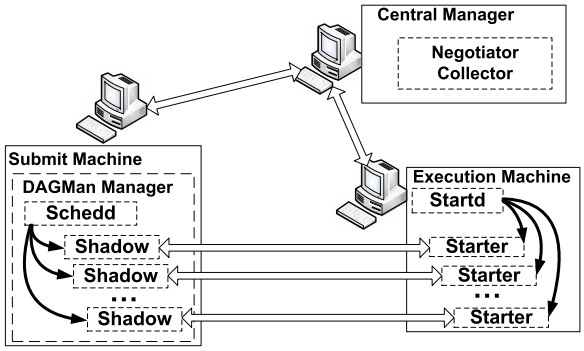

Fig. 4 Condor pool architecture

SWAMP allows users to submit jobs by using a specified repository that stores all input data as well as existing output data. A job can also be submitted using a shared file system to directly access input and output files. In the latter case, the job or module must be able to access the data files from any machine, on which it is deployed through either NFS or AFS. SWAMP provides access to Condor's own native mechanisms for Grid computing as well as other Grid systems, e.g. through Condor-G when Grid universe jobs are sent to Grid resources utilizing Globus software for job execution. The Globus Toolkit provides a framework for building Grid systems and applications.

Figure 4 shows a local Condor pool architecture. Each execution node in the SWAMP condor pool accepts a new job only if the load of the system is not too high and there is enough memory available for efficient execution. The jobs are submitted to the "schedd" process, which stores them in a permanent storage and advertises their needs. The "startd" process on an execution node advertises its resources to the "collector" process. The "negotiator" process regularly fetches these advertisements (ClassAds) from the "collector" and "schedd", and assigns jobs to execution nodes. For every such association, a shadow and a starter process are created and all further communications take place between these two entities.

Figure 5 shows the execution procedure of SWAMP in Grid environments. In order to submit jobs to Grid resources, the submit machine needs to apply for a certificate from a trusted Certificate Authority (CA) for authentication. The submit machine with all Grid software installed and configured manages the jobs running on the Grid. Before running jobs on the Grid, we need to ensure that the submit machine is able to obtain the correct user proxy, which is presented to remote Grid sites during authentication. Condor-G stages data and submits jobs to the Gatekeeper at a remote Grid site. Globus GRAM on the Gatekeeper authenticates users, sends jobs to the local resource manager such as Condor, PBS, and Fork, and notifies users of job statuses. The execution machines run the jobs and notify the local resource managers of job statuses.

Each workflow supported by SWAMP has an output module that runs after all other user modules have completed. The main task of this output module is to send the final output to the Web

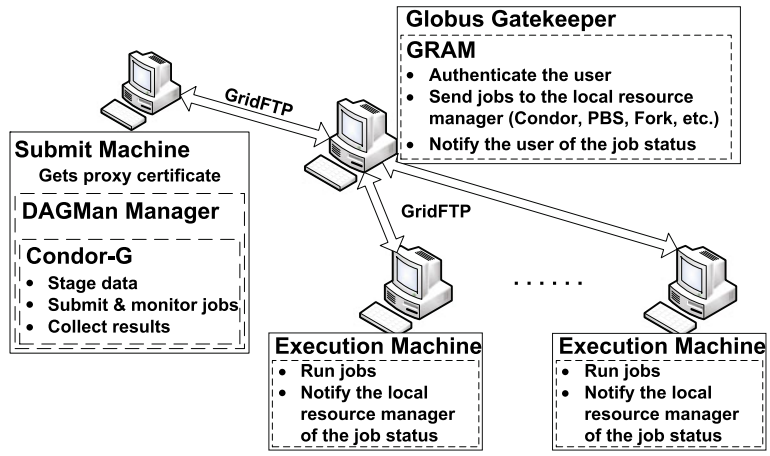

Fig. 5 The execution procedure of SWAMP in Grid environments

Server. Instead of creating a separate daemon, we add this extra job as part of the DAG workflow for final output sending to avoid checking the Condor log files or invoking Condor's management command in polling mode. In Condor batch system, jobs with higher numerical priority run before jobs with lower numerical priority. To send the final output to the user promptly, the output module is assigned with the highest priority.

3.4 Performance Modeling and Prediction

In order to improve the quality of workflow execution, we investigate the problem of modeling scientific computations and predicting their execution time based on a combination of both hardware and software properties. The performance modeling and prediction component queries the resource and provenance database to retrieve the information of computing resources, datasets and executables, or interacts with the resource discovery component to collect such information. In this component, we employ statistical learning techniques to estimate the effective computational power of a given computer node at any time point and compute the total number of CPU cycles needed for executing a given computational program on any input data size. The technical details of this component are provided in Section 4.

3.5 Workflow Mapping

The efficiency of the SWAMP system largely depends on the performance of the mapping schemes that map computing workflows to network nodes in the Condor pool, which is determined by the Condor job dispatch scheme. The mapping scheme currently employed by the Condor scheduler works as a matchmaker of ClassAds. Condor schedules job dispatch by matching the requests for both machine ClassAds and job ClassAds. In addition to the Condor pool, SWAMP is capable of mapping workflows onto a variety of target resources such as those managed by Portable Batch System (PBS) [53] and Load Sharing Facility (LSF) [39], with an improved mapping performance by incorporating a set of well-designed mapping schemes into the Condor negotiator daemon.

The workflow mapping component obtains the information on computing modules such as the executables and data from the resource and provenance database, and then converts an abstract workflow to an executable workflow based on the prediction of the module execution time using the performance modeling and prediction component. The mapping objective is to select an appropriate set of computer nodes in the network and assign each computing module in the workflow to one of those selected nodes to achieve MED for fast response and MFR for smooth data flow. The workflow mapping algorithms are presented in Section 5.

3.6 Resource Discovery

The resource discovery component interacts with an extensive set of existing services and systems to acquire information on data, computing and networking resources, and data movement tools, all of which is stored in the resource and provenance database.

The data discovery process involves two steps: (i) Use data attributes specified by users to query a metadata service, which maps the specified attributes into a list of logical file names. (ii) Use these logical names to query existing registry services such as the Globus Replica Location Service (RLS) [25] to locate the replicas of the required data.

The resource discovery component also communicates with information services and systems to discover available computing resources and corresponding characteristics such as the number of CPUs, CPU frequency, queueing length, and available disk space. Currently supported information services and systems include Monitoring and Discovery Service (MDS) [45], Resource and Site Validation (RSV) [50], and so on.

This component queries a number of resource monitoring services and tools to monitor the status of the networking resources, including perfSONAR [52], Network Weather Service (NWS) [46], One-Way Active Measurement Protocol (OWAMP) [47], and Bandwidth Test Controller (BWCTL) [5].

In order to improve the performance of large data transfer in wide-area networks, the resource

discovery component also queries other information services to locate and employ the available data movement services and tools, including GridFTP [26], Reliable File Transfer (RFT) [57], Storage Resource Broker (SRB) [62], Storage Resource Management (SRM) [63], On-demand Secure Circuit Advance Reservation System (OSCARS) [49], and TeraPaths [67].

3.7 Data Provenance

The data management in heterogeneous networks such as OSG [48] is a key component of the architecture and service of the common infrastructure. Large-scale scientific applications have led to unprecedented levels of data collection, which make it very difficult to keep track of the data generation history. The data provenance component provides information about the derivation history of data starting from its original sources. It enables scientists to verify the correctness of their simulations and reproduce them if necessary. SWAMP provides provenance information related to workflow module execution by analyzing the log files of the Condor system. These log files contain execution information of each module, such as location, time, and output, all of which are stored in the resource and provenance database.

3.8 Resource and Workflow Monitoring Service

The resource and workflow monitoring service uses the callback placed by the workflow execution service to trigger the data provenance component to keep track of the workflow execution progress. It also queries the resource and provenance database to obtain the status information of the computing resources, and then sends the above information on the fly to the workflow display service to update the web interface.

3.9 Workflow Display Service

Each workflow eventually generates an output. The ability to view and manage the output data as soon as it is produced is valuable to users for result interpretation. Users can keep track of the real time information of workflow mapping, execution and result through the web interface. The workflow display service component instantaneously updates the web interface using the information provided by the resource and workflow monitoring service component. The ability to re-dispatch workflows and return the immediate feedback allows the user to modify parameters of any dispatched workflow and re-dispatch individual workflows that require further computation. SWAMP uses coloring convention to indicate the execution status of a module and supports a tabular fashion to show the statistical information of the modules and the links.

4 A Statistical Approach to Performance Modeling and Prediction

Finding a good workflow mapping scheme critically depends on an accurate prediction of the execution time of each individual module in the workflow. The time prediction of a scientific computation does not have a silver bullet as it is determined collectively by several dynamic system factors including concurrent loads, memory size, CPU speed, and also by the complexity of the computational program itself. To tackle the problem of modeling scientific computations and predicting their execution time, we employ a performance model that takes into account both hardware and software properties, which is then used by the prediction method for predicting or estimating the execution time of a given computational job on any computer.

In [18], Dobber et al. provided a broad survey of existing prediction methods and also proposed a Dynamic Exponential Smoothing (DES) method for job runtime prediction on shared processors. All the methods in [18] are solely based on the statistical analysis of historical runtime measurements of user jobs. Although also employing a statistical method, our performance modeling and prediction take a fundamentally different approach using benchmark machines with the main goal to accurately describe the relationship between the runtime of a given scientific computation and various system factors including the hardware properties of the computing node and the software properties of the program. Since

these methods are based on different inputs under different assumptions, their prediction performances may not be directly comparable.

Let vector $H(t)$ represent the hardware profile of a given machine that consists of both static configuration and dynamic usage information at time t, defined as:

$$H(t) = h_{\text{static}} \cup h_{\text{dynamic}}(t) \tag{1}$$

where h_{static} denotes the set of static properties and $h_{\text{dynamic}}(t)$ denotes the set of dynamic properties at time point t. Typically, the hardware specifications of the machine are considered static while the workloads on the machine are dynamic in nature, therefore resulting in a time-varying level of resource capacity. For convenience, we tabulate in Table 1 the notations used in our performance model.

Let C_h represent the *effective processing* (EP) power provided by the available hardware resources of the machine. The EP power determines the number of usable CPU cycles (i.e. the total number of CPU cycles minus those used for system overheads and other background workloads). The relationship between the hardware profile and C_h can be described as:

$$C_h(t) = H(t) \cdot \theta_h + \vartheta_h \tag{2}$$

Table 1 Notations used in the performance model

Symbol	Meaning
$H(t)$	The total hardware resources on the machine in the presence of concurrent workloads at time point t
h_{static}	The set of static hardware properties of the machine
$h_{\text{dynamic}}(t)$	The set of dynamic hardware properties at time point t
S	The set of software properties of the computational job
C_h	The effective processing power
C_s	The total number of CPU cycles required by the computational job
θ_h	The coefficient vector for hardware estimation
θ_s	The coefficient vector for software estimation

where θ_h is the coefficient vector of a regression estimate for hardware profiles, and ϑ_h denotes the set of regression constants.

Now let us consider the software properties of the computational job and estimate the number C_s of CPU cycles that are needed to execute the job. Similarly, we employ a polynomial regression to capture the relationship between C_s and the job's software properties:

$$C_s = S \cdot \theta_s + \vartheta_s \tag{3}$$

where S denotes the vector holding the software properties of the program (computational job), θ_s is the coefficient vector of a regression estimate for software properties, and ϑ_s denotes the set of regression constants.

Given C_h and C_s, the relationship between the execution time and the hardware and software properties is still very complicated. We employ a nonlinear parametric regression analysis method to model such complex behaviors. The estimation of the job execution time $T(t)$ on the machine at time point t is given by:

$$T(t) = f((C_h(t)\theta_1 + C_s\theta_2), \beta) + \vartheta, \tag{4}$$

where θ_1 and θ_2 are the coefficient vectors of a regression estimate for the hardware and software properties, respectively. Equation (4) can be rewritten in the following form:

$$T(t) = f(X_0) \cdot \beta' + \vartheta, \tag{5}$$

where X_0 denotes $C_h(t)\theta_1 + C_s\theta_2$. Equation (5) is linear in terms of the explanatory variable β', and the dimension of the vector class estimator is the same as the dimension of X_0.

For a given computational job running on a given machine, we assume that the available hardware resources $H(t)$ on the machine and the job execution time $T(t)$ on that machine be distributed by a joint probability distribution $P_{H(t),T(t)}$. Note that the software properties of a computational job remain constant, and therefore the total number of CPU cycles required for running the job is fixed for a given input data size. Note that the actual runtime of some algorithms (especially for optimization purposes) may depend on both the input size and the input data. Since our system is focused on scientific applications and the

complexity of scientific computations is largely determined by the dimensional parameters and the number of iterations, our approach is still applicable in most science domains (especially with volume data processing such as visualization),

Now by using the least squares estimate, we calculate the execution time estimation error as:

$$I_{\theta,\vartheta} = \int (\hat{T}(t) - T(t)) dP_{H(t),T(t)}, \tag{6}$$

where $\hat{T}(t)$ denotes the actual amount of time taken. The best estimator for the actual execution time is given by minimizing (6). In order to obtain the best estimation, one would need an accurate distribution $P_{H(t),T(t)}$, which is very hard to find in practice. The problem may not be tractable if this distribution is complex [75].

Let $I^*_{\theta,\vartheta}$ denote the best estimation given by (6):

$$I^*_{\theta,\vartheta} = \min(I_{\theta,\vartheta}). \tag{7}$$

The goal of the estimation is to have a high probability that the estimation error is within some permissible limit ε. Using the result from the sample size estimation [56], we have:

$$P((I_{\theta,\vartheta} - I^*_{\theta,\vartheta}) > \varepsilon) \leq 1 - 8\left(\frac{64e}{\varepsilon}\ln\frac{64e}{\varepsilon}\right)^d e^{-\varepsilon^2\frac{k}{512}}, \tag{8}$$

where k is the sample size.

As mentioned in [75], the same argument holds on the estimation bound. Note that this estimation bound does not depend on the form of distribution $P_{H(t),T(t)}$, but the drawback is that it only ensures the closeness of the estimation error to the best possible linear approximation. It is possible that the latter itself may not be unsatisfactory if the underlying relationship is non-linear.

The factors that affect the job execution time can be categorized into hardware and software parameters. Since the hardware parameters themselves are an exhaustive list, we consider the most significant hardware properties as shown in Table 2.

The size of used RAM provides important information on the amount of concurrent workloads in the system as a process always consumes a certain portion of RAM. The buffer and cache size largely affects the performance of I/O operations. Since the total execution time of the program consists of both computation time and I/O time, it is necessary to consider these parameters collectively to obtain a fair estimate of the effective processing power of the machine. These parameters can be collected by some system tools such as the *free* command in Linux. We take Whetstone [68] as one of the hardware parameters instead of the CPU clock speed (frequency) because the number of floating point arithmetic operations that can be done per second by the processor is dependent on both the clock speed, processor architecture, and the concurrent load on the machine.

Table 2 Hardware properties used for prediction

Hardware properties
Size of used and free RAM
Size of used and free buffer Size of used and free cache
Whetstone measurement

Besides the hardware properties, the job execution time also depends on the computational complexity of the program, which is hard to estimate just by examining the source code. Since the program might use multiple system or user-defined libraries, obtaining a holistic computational complexity function is a very challenging task. In order to address this issue, we use TPROF tool [51] to calculate the number of CPU cycles used by the module. We repeat this job execution experiment with different input data sizes on a benchmark machine and perform a polynomial regression to fit a curve among these measured data points. The fitted curve is of the form:

$$f(z) = a_0 + a_1 \cdot z + a_2 \cdot z^2 + \ldots + a_n \cdot z^n, \tag{9}$$

where z is the data size of the input. The fitted curve is then used to estimate computational complexity of the program. This complexity estimator provides us an expected number of CPU cycles to execute the computational program on any given input data size, which is used as one of the independent variables in the prediction model. Since the number of CPU cycles depends on the processor architecture, to be more specific, the microinstruction set, the estimated program complexity

needs to be adjusted appropriately on a machine with a different architecture, for example, by calculating and applying the ratio of the required CPU cycles for performing the same floating-point operation (flop) between the current machine and the benchmark machine. The prediction model is built using a multivariate nonlinear regression function which takes estimated CPU cycles, and hardware parameters shown in Table 2 as independent variables and estimates the amount of time module takes on the machine. The prediction model is of the form:

$$T = \frac{A(f(z))}{B(I)}, \tag{10}$$

where T is the dependent variable which represents time taken by the executable, I represents the independent variables. Once its regression form is determined, $f(z)$ produces a concrete value for a given input data size z. However, directly using this value (the estimated number of CPU cycles) may not yield an accurate time prediction because the actual number of CPU cycles allocated to a job execution may be subject to a certain variation, for example, due to frequent page swapping if the program requires more memory space than what is available. The function $A(\cdot)$ is in the form of $A(x) = p_0 + p_1 \cdot x^{k_0} + p_2 \cdot x^{k_1} + ... + p_n \cdot x^{k_{n-1}}$ where $p_0....p_n, k_0...k_n$ are real numbers, and the purpose of using this function is to capture such variations. Similarly, we use a polynomial function $B(\cdot)$ formed by taking all possible combinations of the independent variables I to capture the effect of each independent variable when acting alone and in combination of other independent variables. In the simplest case with two dependent variables, x_1 and x_2, a possible form of the first order the function $B(\cdot)$ could take is $B(x_1, x_2) = p_0 + p_1 \cdot x_1 + p_2 \cdot v_2 + p_3 \cdot (v_1 \cdot v_2)$.

The estimated computational complexity of the module using polynomial regression in (9) does not indicate the inherent parallelism of the module.

Once both hardware and software properties are measured or calculated through either system tools or regression-based estimators, we combine them to achieve a good estimate of the job execution time based on regression techniques, as shown in Fig. 6.

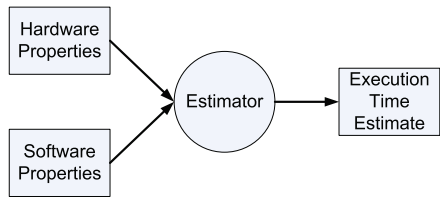

Fig. 6 Regression-based performance prediction considering both hardware and software properties

5 Workflow Mapping Optimization

The SWAMP system incorporates a workflow mapper, which consists of a class of rigorously designed mapping algorithms for end-to-end workflow performance optimization.

5.1 Problem Formulation

The workflow is modeled as a Directed Acyclic Graph (DAG) $G_w = (V_w, E_w)$, $|V_w| = m$, where vertices represent computing modules starting from module w_0 and ending at module w_{m-1}. The dependency between a pair of adjacent modules w_i and w_j is represented by a directed edge $e_{i,j} \in E_w$ between them. Module w_j receives a data input from each of its preceding modules and performs a predefined computing routine whose complexity is modeled as a function $\lambda_{w_j}(\cdot)$ on the aggregate input data size z_{w_j}. The overlay computer network is modeled as an arbitrary weighted graph $G_c = (V_c, E_c)$, consisting of $|V_c| = n$ computer nodes interconnected by $|E_c|$ overlay network links. Following the modeling approach in Section 4, we use a normalized variable p_i to represent the overall effective processing power of node v_i without specifying its detailed system resources. The link $l_{i,j}$ between nodes v_i and v_j has Bandwidth (BW) $b_{i,j}$ and Minimum Link Delay (MLD) $d_{i,j}$. We specify a pair of source and destination nodes (v_s, v_d) to run the start module w_0 and the end module w_{m-1}, respectively, and further assume that module w_0 serves as a data source without any computation and module w_{m-1} performs a terminal task without any data transfer.

We use $T_{\text{exec}}(w, v) = \sum \frac{\alpha(t) \cdot \delta_w(t)}{p}$ to denote the execution time of module w on node v, where $\alpha(t)$ is the number of concurrent modules on node v

during Δt, $\delta_w(t) = \frac{p}{\alpha(t)}\Delta t$ denotes the amount of partial module execution completed during time interval $[t,\ t+\Delta t]$ when $\alpha(t)$ remains unchanged, and $\lambda_w(z_w) = \sum \delta_w(t)$ is the total computational requirement of module w. We use $T_{\text{tran}}(e,l) = \sum \frac{\beta(t)\cdot\delta_e(t)}{b} + d$ to denote the data transfer time of dependency edge e over network link l, where $\beta(t)$ is the number of concurrent data transfers over link l during Δt, $\delta_e(t) = \frac{b}{\beta(t)}\Delta t$ denotes the amount of partial data transfer completed during time interval $[t,\ t+\Delta t]$ when $\beta(t)$ remains unchanged, and $z_e = \sum \delta_e(t)$ is the total data transfer size of dependency edge e. Note that computing and networking resources could be shared by multiple modules and edges that are processing and transferring different input datasets. Due to the dynamics in concurrent workload on nodes and concurrent traffic over links, both $\alpha(t)$ and $\beta(t)$ are time-varying in nature.

Based on the analytical cost models, we consider two performance metrics:

(i) *End-to-end Delay* (ED), which is the total completion time of the entire workflow from the time when a single dataset is fed into the first module to the time when the final result is generated at the last module. Once a mapping scheme is determined, the ED is calculated as the total time cost incurred on the critical path (CP), i.e. the longest path. We denote the set of contiguous modules on the CP that are allocated to the same node as a "group", and refer to those modules located on the CP as "critical" modules and others as "branch" or "non-critical" modules. A general mapping scheme divides the CP into q $(1 \le q \le m)$ contiguous groups $g_i, i = 0, 1, \cdots, q-1$ of critical modules and maps them to a network path P of not necessarily distinct q nodes, $v_{P[0]}, v_{P[1]}, \cdots, v_{P[q-1]}$ from source $v_s = v_{P[0]}$ to destination $v_d = v_{P[q-1]}$. The ED of a mapped workflow is defined as:

$$
\begin{aligned}
&T_{\text{ED}}\left(\text{CP mapped to a path } P \text{ of } q \text{ nodes}\right) \\
&= T_{\text{exec}} + T_{\text{tran}} = \sum_{i=0}^{q-1} T_{g_i} + \sum_{i=0}^{q-2} T_{e(g_i,g_{i+1})} \\
&= \sum_{i=0}^{q-1}\left(\sum_{j\in g_i,\, j\ge 1}\left(\sum \frac{\alpha_j(t)\cdot\delta_{w_j}(t)}{p_{P[i]}}\right)\right) \\
&\quad + \sum_{i=0}^{q-2}\left(\sum \frac{\beta_{e(g_i,g_{i+1})}(t)\cdot\delta_{e(g_i,g_{i+1})}(t)}{b_{P[i],P[i+1]}} + d_{P[i],P[i+1]}\right),
\end{aligned}
\tag{11}
$$

where $e(g_i, g_{i+1})$ denotes the edge from group g_i to g_{i+1} mapped to link $l_{P[i],P[i+1]}$ between nodes $v_{P[i]}$ and $v_{P[i+1]}$.

(ii) *Frame Rate* (FR) or throughput, i.e. the inverse of the global Bottleneck Time (BT) of the workflow. The bottleneck time T_{BT} could be either on a node or over a link, defined as:

$$
\begin{aligned}
&T_{\text{BT}}(G_w \text{ mapped to } G_c) \\
&= \max_{\substack{w_i\in V_w, e_{j,k}\in E_w \\ v_{i'}\in V_c, l_{j',k'}\in E_c}}\begin{pmatrix} T_{\text{exec}}(w_i, v_{i'}), \\ T_{\text{tran}}(e_{j,k}, l_{j',k'}) \end{pmatrix} \\
&= \max_{\substack{w_i\in V_w, e_{j,k}\in E_w \\ v_{i'}\in V_c, l_{j',k'}\in E_c}}\left(\begin{matrix} \sum \frac{\alpha_i(t)\cdot\delta_{w_i}(t)}{p_{i'}}, \\ \sum \frac{\beta_{j,k}(t)\cdot\delta_{e_{j,k}}(t)}{b_{j',k'}} + d_{j',k'} \end{matrix}\right).
\end{aligned}
\tag{12}
$$

We assume that the inter-module communication cost on the same node is negligible.

Since the execution start time of a module depends on the availability of its input datasets, the modules assigned to the same node may not run simultaneously. The same is also true for concurrent data transfers over the same network link. The workflow mapping problem with arbitrary node reuse for MED or MFR is formally defined as follows: *Given a DAG-structured computing workflow $G_w = (V_w, E_w)$ and a heterogeneous computer network $G_c = (V_c, E_c)$, we wish to find a mapping scheme that assigns each computing module to a network node such that the mapped workflow achieves*:

$$
\text{MED} = \min_{\text{all possible mappings}} \left(T_{\text{ED}}\right), \tag{13}
$$

$$
\text{MFR} = \max_{\text{all possible mappings}} \left(\frac{1}{T_{\text{BT}}}\right). \tag{14}
$$

Note that MED is the minimum sum of time costs along the CP while MFR is achieved by identifying and minimizing the time T_{BT} on a global bottleneck node or link among all possible mappings. We maximize FR to produce the smoothest data flow in streaming applications when multiple datasets are continuously generated and fed into the workflow.

5.2 Design of Mapping Algorithms

A workflow mapping scheme must account for the temporal constraints in the form of execution order of computing modules and spatial constraints in the form of geographical distribution of network nodes and their connectivity. The general DAG mapping problem is known to be NP-complete [2, 22, 35, 38] even on two processors without any topology or connectivity restrictions [1], which rules out any polynomial optimal solutions. We incorporate two efficient workflow mapping algorithms into the SWAMP system to optimize workflow performance in terms of MED and MFR.

For unitary processing applications, we incorporate into our system an improved *Recursive Critical Path* (impRCP) algorithm [74], which recursively chooses the CP based on the previous round of calculation and maps it to the network using a dynamic programming (DP)-based procedure until the mapping results converge to an optimal or suboptimal point. For the sake of completeness, we provide a brief summary of those key steps in the impRCP algorithm:

1. Assume the network topology to be complete with identical computing nodes and communication links, and determine the initial computing/transfer time cost components.
2. Find the CP of the workflow with initial time cost components using a longest path algorithm.
3. Remove the assumption on resource homogeneity and connectivity completeness, and map the current CP to the actual network using the optimal pipeline mapping algorithm based on a DP-based procedure for MED with arbitrary node reuse as proposed in [29].
4. Map non-critical modules using A^* and Beam Search algorithms.
5. Compute a new CP using updated time cost components and calculate the new MED.

Steps 2–5 are repeated until a certain condition is met, for example, the difference between a new MED resulted from the current mapping and an old MED resulted from the previous mapping is less than a preset threshold. The complexity of impRCP is $O(\rho \cdot m \cdot n \cdot |E_w|)$, where ρ is the predefined beam width in the BS algorithm, m and $|E_w|$ are the number of modules and edges in the workflow, respectively, and n is the number of nodes in the network.

For streaming applications with serial data inputs, we incorporate into our system a *Layer-oriented Dynamic Programming* algorithm [27], referred to as LDP, which maps workflows to networks by identifying and minimizing the global BT to achieve the smoothest dataflow. The key idea of LDP is to first sort a DAG-structured workflow in a topological order and then map computing modules to network nodes on a layer-by-layer basis, while taking into consideration both module dependency in the workflow and node connectivity in the network. We use a two-dimensional DP table to record the intermediate mapping result at each step where a module is mapped layer-by-layer to a strategically selected network node, where the horizontal coordinates represent the sequence numbers of topologically sorted modules, and the vertical coordinates represent the labels of nodes starting from the source node v_s to the destination node v_{n-1}.

To reduce the search complexity, we propose a *Greedy LDP*, which selects the best node in each column that yields the minimum global BT for mapping the current module, hence filling one cell only in each column. In Greedy LDP, we recursively calculate the minimum BT of the sub-solution that maps the subgraph (a partial workflow) consisting of the current module w_j and all the modules before the current layer to the network until the last module w_{m-1} is reached and mapped to the destination v_d, which is represented by the right bottom cell in the DP table. The complexity of this algorithm is $O(l \cdot n \cdot |E_w|)$,

where l is the number of layers in the topologically sorted workflow, n is the number of nodes in the network, and $|E_w|$ is the number of edges in the workflow.

6 System Implementation and Evaluation

We conduct extensive comparative performance evaluations based on both simulations and experiments through system implementation and real deployment to illustrate the performance superiority of the proposed solution over other workflow mapping algorithms and the Condor default execution model.

6.1 Simulation-based Performance Evaluation

The proposed impRCP and Greedy LDP mapping algorithms are implemented in C++. For performance comparison, we also implement three existing algorithms, namely (i) *Greedy* A^* [58], which maps subtasks onto a large number of sensor nodes based on A^* algorithm, whose complexity is $O(m^2 + kn(m-k))$, where m is the number of modules, n is the number of sensor nodes, and k is the number of modules in the independent set; (ii) *Streamline* (SL) [2], which maximizes the throughput of an application by assigning the best resources to the most needy stages, whose complexity is $O(mn^2)$, where m is the number of stages or modules in the dataflow graph and n is the number of resources or nodes in the network; and (iii) *Greedy*, which employs a simple local optimal search procedure at each mapping step, whose complexity is $O(|E_w| \cdot |E_c|)$, where $|E_w|$ is the number of edges in the workflow and $|E_c|$ is the number of links in the network. We conduct an extensive set of mapping experiments for both MED and MFR using simulation datasets generated by randomly varying the parameters of workflows and networks within a suitably selected range of values, represented by a four-tuple (m, $|E_w|$, n, $|E_c|$): m modules and $|E_w|$ edges in the computing workflow, and n nodes and $|E_c|$ links in the computer network.

The MED measurements of 20 problem sizes indexed from 1 to 20 are tabulated in Table 3 and visualized in Fig. 7, respectively. In each problem size, we generate ten random mapping instances and calculate the average performance measurement. The percentage enclosed in the

Table 3 MED measurements of four algorithms in comparison

Problem index	Problem size $m, \|E_w\|, n, \|E_c\|$	MED (s) impRCP	Greedy A^*	SL	Greedy
1	4, 6, 6, 29	0.50	0.52 (3.8 %)	0.51 (2.0 %)	0.52 (3.8 %)
2	6, 10, 10, 86	1.05	1.19 (11.8 %)	1.14 (7.9 %)	1.11 (5.4 %)
3	10, 18, 15, 207	1.15	1.61 (28.6 %)	1.65 (30.3 %)	1.94 (40.7 %)
4	13, 24, 20, 376	2.15	2.41 (10.8 %)	2.45 (12.2 %)	2.33 (7.7 %)
5	15, 30, 25, 597	2.23	2.88 (22.6 %)	2.94 (24.1 %)	2.87 (22.3 %)
6	19, 36, 28, 753	1.96	2.71 (27.7 %)	2.72 (27.9 %)	3.03 (35.3 %)
7	22, 44, 31, 927	2.46	2.63 (6.5 %)	2.76 (10.9 %)	2.93 (16.0 %)
8	26, 50, 35, 1180	1.85	2.79 (33.7 %)	4.00 (53.8 %)	2.56 (27.7 %)
9	30, 62, 40, 1558	3.56	5.06 (29.6 %)	5.22 (31.8 %)	4.45 (20.0 %)
10	35, 70, 45, 1963	4.05	4.37 (7.3 %)	5.79 (30.1 %)	7.02 (42.3 %)
11	38, 73, 47, 2153	4.62	5.78 (20.1 %)	6.89 (32.9 %)	6.92 (33.2 %)
12	40, 78, 50, 2428	2.01	2.97 (32.3 %)	3.13 (35.8 %)	2.71 (25.8 %)
13	45, 96, 60, 3520	3.64	5.13 (29.0 %)	5.92 (38.5 %)	5.17 (29.6 %)
14	50, 102, 65, 4155	2.80	5.12 (45.3 %)	4.37 (35.9 %)	4.82 (41.9 %)
15	55, 124, 70, 4820	4.49	6.86 (34.6 %)	6.63 (32.3 %)	7.46 (39.8 %)
16	60, 240, 75, 5540	8.67	9.30 (6.8 %)	14.63 (40.7 %)	13.43 (35.4 %)
17	75, 369, 90, 7990	15.41	18.25 (15.6 %)	20.53 (24.9 %)	17.92 (14.0 %)
18	80, 420, 100, 9896	19.81	25.04 (20.9 %)	26.20 (24.4 %)	22.71 (12.8 %)
19	90, 500, 150, 22346	24.01	28.83 (16.7 %)	28.18 (14.8 %)	25.84 (7.1 %)
20	100, 660, 200, 39790	24.79	34.71 (28.6 %)	29.78 (16.8 %)	26.62 (6.9 %)

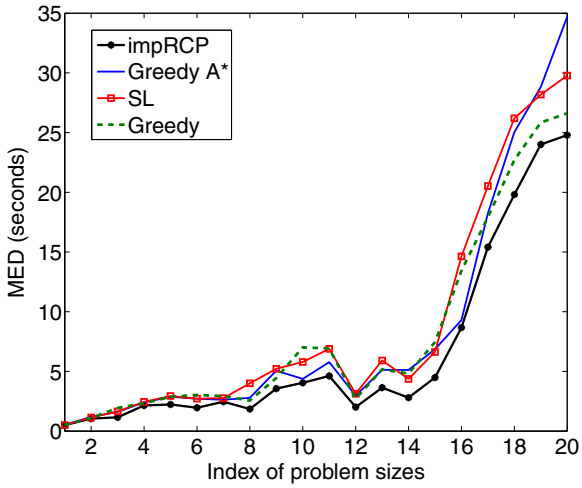

Fig. 7 MED performance comparison among four algorithms

parentheses after each average measurement in the table is the MED improvement of impRCP over the other algorithms in comparison, defined as $|\frac{\text{MED}_{\text{others}}-\text{MED}_{\text{impRCP}}}{\text{MED}_{\text{others}}}| \times 100$ %. We observe that the impRCP algorithm consistently outperforms all other methods. These performance measurements also illustrate a trend that the superiority of the proposed mapping approach becomes more obvious as the problem size increases.

We also compare the MFR performance of the proposed Greedy LDP algorithm with that of the Greedy A^*, SL, and Greedy algorithms. The MFR measurements in 20 mapping problems of various sizes indexed from one to 20 are tabulated in Table 4. Similarly, these are the average measurements calculated from ten instances in each problem size and the performance improvement in the percentage term is enclosed for each average measurement. We observe that Greedy LDP outperforms the other methods in all the cases we studied. Note that these algorithms may produce the same optimal mapping results in small-scale problem cases. Another significant advantage of Greedy LDP is that it considers both module dependency and network connectivity during workflow mapping, therefore leading to a much higher success rate in mapping (no invalid mapping in all the cases we studied). Since both Greedy A^* and SL are designed for complete networks without considering any topology constraint on mapping, they may miss certain feasible

Table 4 MFR measurements of four algorithms in comparison

Problem	Problem size	MFR (frames/s)			
index	$m, \|E_w\|, n, \|E_c\|$	Greedy LDP	Greedy A^*	SL	Greedy
1	4, 6, 6, 29	1.025	0.980 (4.6 %)	0.980 (4.6 %)	0.980 (4.6 %)
2	6, 10, 10,86	1.854	1.201 (54.4 %)	1.201 (54.4 %)	1.201 (54.4 %)
3	10, 18, 15, 207	1.544	1.325 (16.5 %)	1.072 (44.0 %)	1.325 (16.5 %)
4	13, 24, 20, 376	1.716	–	1.048 (63.7 %)	–
5	15, 30, 25, 597	1.724	1.259 (36.9 %)	0.684 (152.0 %)	1.154 (49.4 %)
6	19, 36, 28, 753	1.116	1.050 (6.3 %)	0.668 (67.1 %)	–
7	22, 44, 31, 927	1.692	–	1.164 (45.4 %)	1.674 (1.1 %)
8	26, 50, 35, 1180	0.946	0.774 (22.2 %)	0.734 (28.9 %)	0.801 (18.1 %)
9	30, 62, 40, 1558	1.393	0.972 (43.3 %)	0.780 (78.6 %)	0.736 (89.3 %)
10	35, 70, 45, 1963	0.812	–	–	0.677 (19.9 %)
11	38, 73, 47, 2153	0.863	0.683 (26.4 %)	0.502 (71.9 %)	0.675 (27.9 %)
12	40, 78, 50, 2428	1.461	1.230 (18.8 %)	0.730 (100.1 %)	1.217 (20.0 %)
13	45, 96, 60, 3520	0.885	0.800 (10.6 %)	–	0.752 (17.7 %)
14	50, 102, 65, 4155	1.911	1.145 (66.9 %)	1.191 (60.5 %)	0.774 (146.9 %)
15	55, 124, 70, 4820	0.836	0.593 (41.0 %)	0.569 (46.9 %)	–
16	60, 240, 75, 5540	0.521	0.392 (32.9 %)	0.391 (33.2 %)	0.396 (31.6 %)
17	75, 369, 90, 7990	0.373	0.238 (56.7 %)	–	0.241 (54.8 %)
18	80, 420, 100, 9896	0.333	0.205 (62.4 %)	0.220 (51.4 %)	0.205 (62.4 %)
19	90, 500, 150, 22346	0.417	0.278 (50.0 %)	0.318 (31.1 %)	0.364 (14.6 %)
20	100, 660, 200, 39790	0.317	0.187 (69.5 %)	0.190 (66.8 %)	0.263 (20.5 %)

mapping solutions when two dependent modules are mapped to two nonadjacent nodes. This problem can also occur in the Greedy algorithm when the last selected node does not have a link to the destination node. In Table 4, we cross out the MFR measurement in an invalid mapping scheme. For a visual comparison, we remove those invalid mapping results and re-compute them using new randomly generated instances with the same problem size. We plot the MFR measurements produced by these four algorithms in Fig. 8, which shows that Greedy LDP consistently exhibits comparable or superior MFR performances over the other three algorithms.

Since the MED represents the total delay from source to destination, a larger problem size with more network nodes and computing modules generally (not absolutely, though) incurs a longer mapping path resulting in a longer ED, which explains the increasing trend in Fig. 7. The MFR, the reciprocal of the global bottleneck in a mapped workflow, is not particularly related to the problem size, and hence the performance curves lack an obvious increasing or decreasing trend in response to varying problem sizes. The slightly decreasing trend in Fig. 8 is due to the fact that larger problem sizes are more likely to have one node or one link shared by more module executions or data transfers.

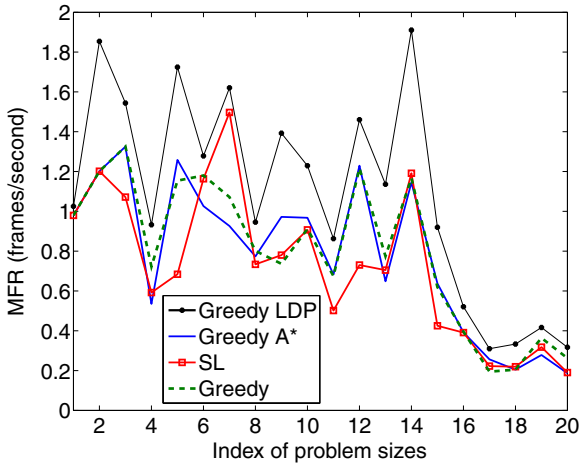

Fig. 8 MFR performance comparison among four algorithms

6.2 Experimental Results on Climate Modeling Workflows

6.2.1 A Brief Introduction to NCAR Single Column CAM (SCAM)

Numerical modeling of the climate system and its sensitivity to anthropogenic forcing is a highly complex scientific problem. The progress towards accurately representing the climate system using global numerical models is primarily paced by uncertainties in the representation of non-resolvable physical processes, most often treated by what is known as physical parameterization. Evaluating many parametric approaches that attempt to represent the physical processes in atmospheric models can be both scientifically complex and computationally expensive.

Because of the high computational expense associated with evaluating new parameterizations using a complete atmospheric general circulation model, a highly flexible and computationally inexpensive single column modeling environment for the investigation of parameterized physics targeted for global climate models has been developed. In particular, this framework is designed to facilitate the development and evaluation of physical parameterizations for the NCAR Community Atmosphere Model (CAM). The SCM modeling environment provides a framework for making initial assessments of physical parameterization packages and allows for the incorporation of both in-situ forcing datasets (e.g., Atmospheric

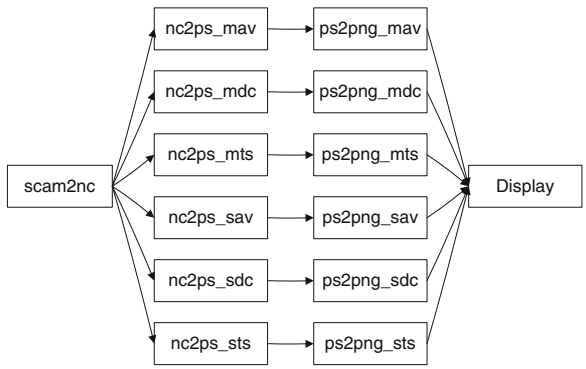

Fig. 9 SCAM workflow structure

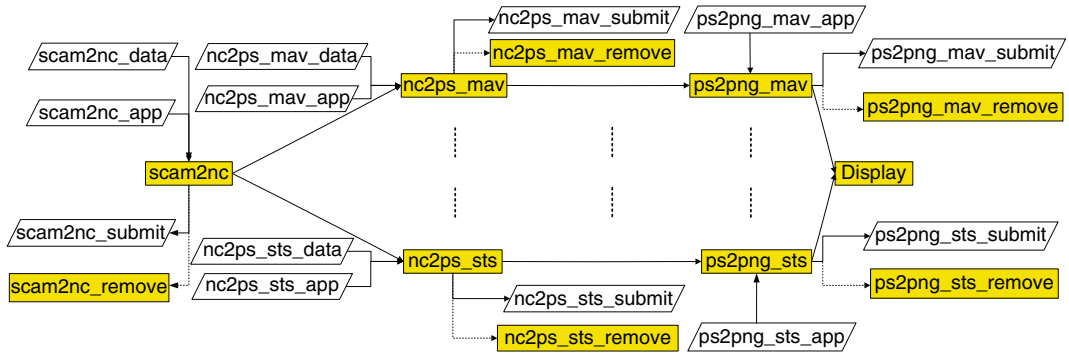

Fig. 10 SCAM executable workflow structure

Radiation Measurement (ARM) data) and synthetic, user-specified, and forcing datasets, to help guide the refinement of parameterizations under development. Diagnostic data that are used to evaluate model performance can also be trivially incorporated. The computational design of the SCM framework allows assessments of both the scientific and computational aspects of the physics parameterizations for the NCAR CAM because the coding structures at the physics module levels are identical. This framework has widespread utilities and helps enrich the pool of researchers working on the problem of physical parameterization since few have access to or can afford testing new approaches in atmospheric GCMs. Another strength of this approach is that it provides a

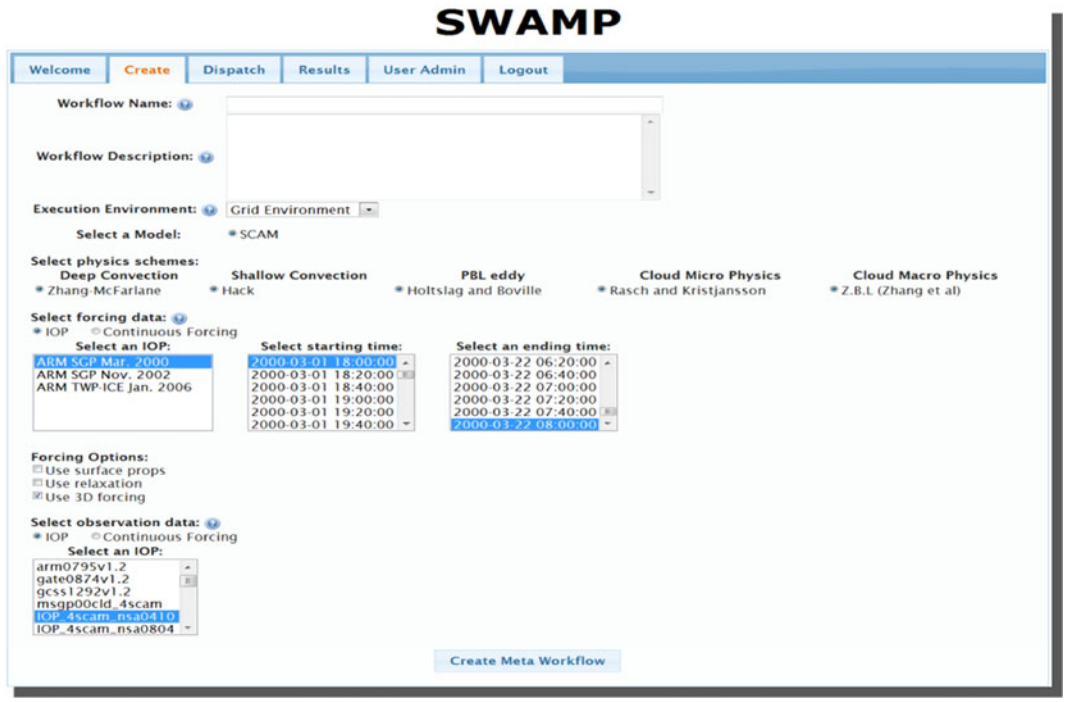

Fig. 11 SCAM workflow generation

common framework to investigate the scientific requirements for successful parameterization of subgrid-scale processes.

6.2.2 SCAM Workflow Structure

As shown in Fig. 9, we represent SCAM as a workflow with 14 modules to be executed using SWAMP in Grid environments with GridFTP-enabled inter-module data transfer. For each combination of available physics packages, data to be used, and experiment controls, we treat it as a separate dispatch of one SCAM workflow. There are a dynamic number of workflow dispatches for the SCAM experiment depending on different combinations.

Figure 10 shows the corresponding executable SCAM workflow structure where a solid line represents a data flow and a dotted line represents a control flow. Note that several auxiliary modules are added to the SCAM workflow to stage in data, executable, and libraries required by the workflow modules and remove the data used or produced by these modules at the remote Grid sites.

6.2.3 SCAM Workflow Generation and Customization

Given a combination of available physics packages, data to be used, and experiment controls, a SCAM model experiment (i.e. scam2nc module) is performed to generate simulation data. The physics packages available in SWAMP include parameterizations for convection, turbulence, and cloud/precipitations. The used data include initial and boundary conditions, so-called large-scale forcing that would drive the responses from the physics processes but are considered independent

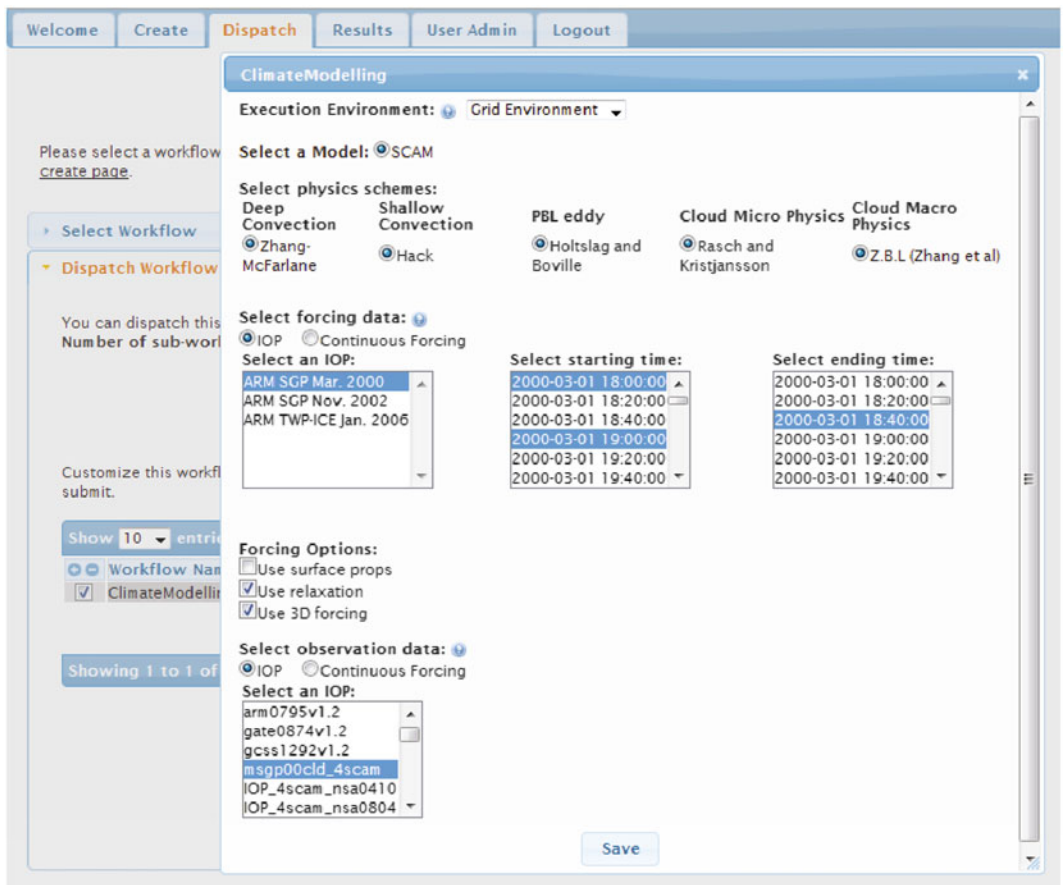

Fig. 12 SCAM workflow customization

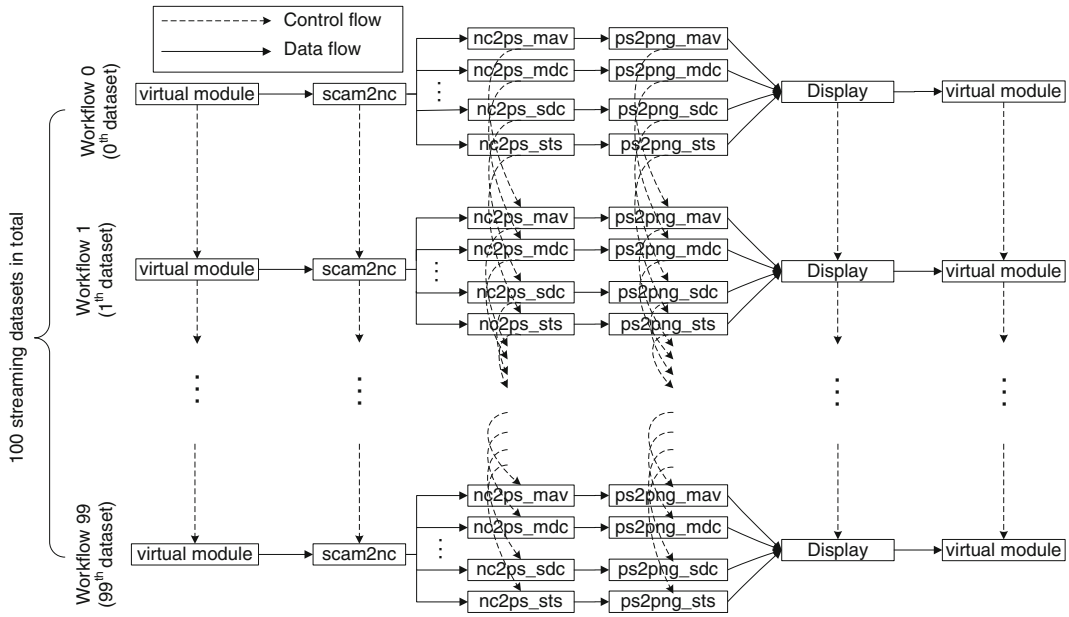

Fig. 13 The virtual SCAM workflow structure for streaming processing

of the physics. These data are in general carefully constructed to minimize uncertainties due to other factors not included in the intended parameterizations and isolate the influence just due to the parameterized processes. The experiment controls include single case targeting a specific event, a set of similar events, ensemble experiments to account for uncertainties in input data, and simultaneous multi-regime evaluations. Both model simulated data and observed data are needed for post-processing and visualization. The SCAM workflow uses csh script as a master control, NCAR Command Language (NCL) script to process data and produce plots, and Ghostscript to convert the postscript image format to web-ready format (e.g. jpeg, png).

The combinations of available physics packages, data to be used and experiment controls require a number of workflow dispatches to extensively evaluate new parameterizations. We developed a separate program to automatically generate DAG and submit files for a climate modeling workflow. As shown in Fig. 11, all the combinations of parameters are gathered from web input and passed into SCAM model through a startup file.

Once the workflow meta file is generated, one SCAM workflow will be dispatched multiple times immediately in a batch mode and the user can customize the SCAM workflow before dispatch, as shown in Fig. 12.

6.2.4 Experimental Settings and Performance Comparison

For performance evaluation, we set up a distributed heterogeneous network testbed consisting of 9 PC workstations located at University of Memphis and three PC workstations at Southern Illinois University at Carbondale. These computers have different hardware configurations in terms of CPU frequency, memory size, and disk space. The most important parameter, CPU frequency, varies from 1.2 GHz to 3.4 GHz. We create an arbitrary network topology by configuring different firewall settings and measure the

Table 5 Input data sizes in ten SCAM experiments

Case Idx	1	2	3	4	5	6	7	8	9	10
# of days	2	4	6	8	10	12	14	16	18	20

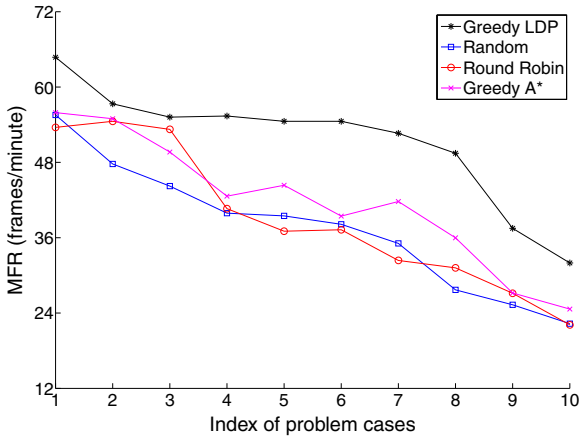

Fig. 14 MFR comparison between Greedy LDP, Random, Round Robin, and Greedy A^*

link bandwidth between two adjacent computers using the "iperf" tool [23]. We deploy SWAMP in this network testbed.

Since Condor is not inherently capable of supporting streaming processing of multiple continuous datasets, for MFR experiments, we link 100 SCAM workflows together to form a virtual workflow as shown in Fig. 13, where each module in the previous component workflow is connected to its corresponding module in the succeeding component workflow with a virtual control flow edge denoted by a dashed arrow. When the current module finishes executing the current dataset, it sends the output along the horizontal data flow edge (a solid arrow) to its succeeding module in the same workflow, and also sends a signal through the vertical control flow edge to the corresponding module in the succeeding component workflow. When the corresponding module receives the signal, it starts executing the next dataset immediately, and the same is applied to the rest of the modules.

The SCAM workflows use the Intensive Observing Period (IOP) data in March 2000, which provide transient forcing information to SCAM physics, where the data source is collected from observational field programs such as GATE, ARM, and TOGA COARE. To obtain an accurate performance estimation, we run each SCAM module with a certain input data size z on every computer node and measure its execution time t. The time cost coefficient c per data unit for the module on that particular node is calculated as $c = t/z$, which is stored in a 2D table with rows representing modules and columns representing computer nodes. Each module execution is repeated for ten times on every node and the average execution time is used to compute c.

Fig. 15 SCAM execution on OSG using SWAMP

Since there are 20 days of IOP data available in March 2000, we choose a different number of days in each experiment to control the input data size. Table 5 lists the number of days used in ten SCAM workflow experiments.

We run SCAM streaming experiments using the data sizes in Table 5 to compare the MFR measurements of the proposed Greedy LDP algorithm with those of the *Random*, *Round Robin*, and Greedy A^* workflow mapping algorithms. The *Random* algorithm assigns each SCAM module to one of the 12 computers on a random basis while the *Round Robin* algorithm assigns SCAM modules to the 12 computers in a round-robin manner. The performance curves produced by these four algorithms are plotted in Fig. 14. We observe that Greedy LDP outperforms the other three algorithms in all the cases we studied. Since we use the same workflow structure and mapping scheme for different input data sizes (which is different from the simulation setting), a larger data size is more likely to have a longer execution time, resulting in a smaller MFR, which explains the decreasing trend in each curve. We do not have the performance comparison in the OSG environment because the resource information required by the mapping schemes are not available on OSG, which does not provide detailed node information within each Grid site but only the high-level site information via information services.

6.3 A Real-life Use Case: Climate Modeling Workflow on Open Science Grid (OSG)

6.3.1 SCAM Workflow Execution on OSG

SWAMP provides a web-based interface to automate and manage the SCAM workflow execution and uses a special site-level workflow

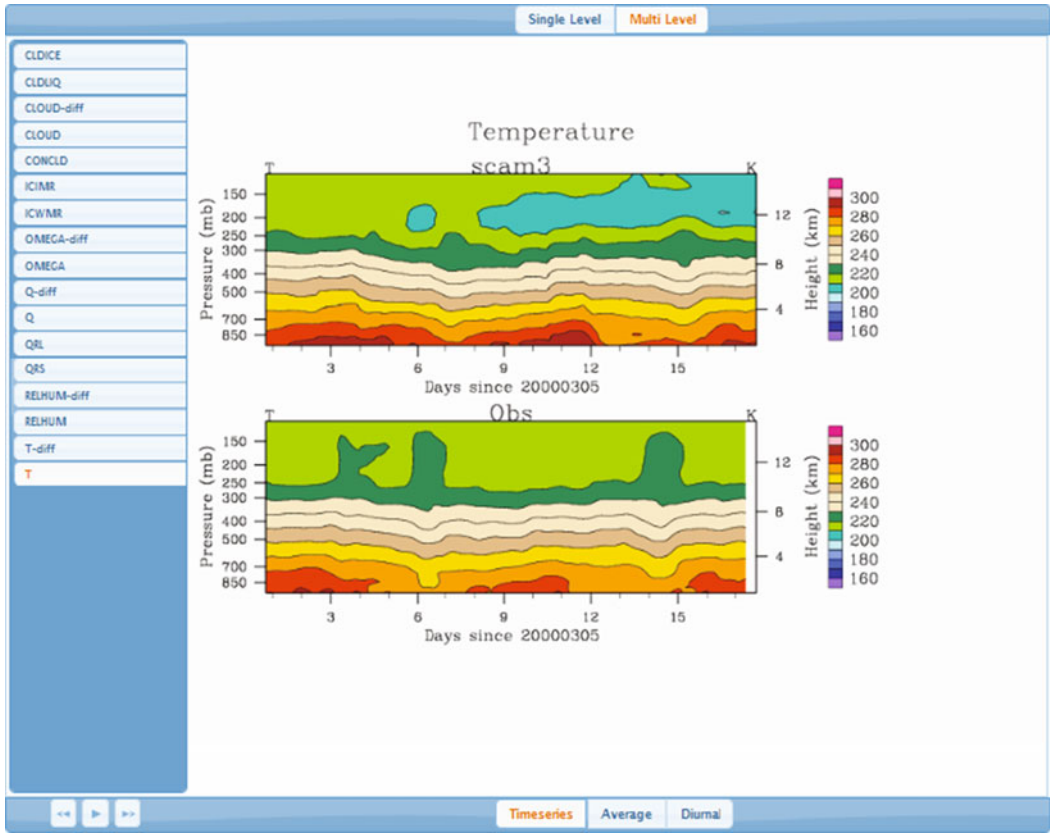

Fig. 16 A gallery of final images generated by the SCAM workflow executed on OSG using SWAMP

mapper to optimize its end-to-end performance on OSG.

Figure 15 shows an example mapping of the SCAM workflow onto three OSG sites. In this example, DAGMan Manager and Condor-G are installed on the submit machine belonging to engagement virtual organization at University of North Carolina. Condor-G stages in the executables, libraries and data of all the modules to the corresponding Grid sites in parallel using GridFTP and submits the first scam2nc module to RENCI site at University of North Carolina. After the scam2nc module completes, Condor-G stages out its output to the Nebraska Omaha site at University of Nebraska using GridFTP and submits all the nc2ps modules to this site. At the same time when all the nc2ps modules are running on the Nebraska Omaha site, SWAMP runs scam2nc_submit and scam2nc_remove jobs to transfer the output of scam2nc back to the submit machine in order to keep track of the data provenance and remove all the data used or produced by scam2nc. When any of the nc2ps modules finishes execution, Condor-G stages out its output to the Brookhaven Atlas Tier1 site at Brookhaven National Laboratory (BNL) and submits the corresponding ps2png module to this site. At the same time when the ps2png modules are running on the Brookhaven Atlas Tier1 site, the SWAMP system runs corresponding nc2ps_submit and nc2ps_remove jobs to transfer the output of nc2ps modules back to the submit machine in order to keep track of the data provenance and remove all the data used or produced by nc2ps modules. Condor-G stages out the data from BNL to the Web Server at University of Memphis using GridFTP for display and also sends those png files back to the submit machine for data provenance.

6.3.2 Display of SCAM Workflow Results

Each SCAM workflow eventually generates a number of images. A gallery of final images for one SCAM workflow are provided on the web interface for a visual examination, as shown in Fig. 16. Also, based on the displayed results, SWAMP enables the user to customize those steerable computing modules of the SCAM workflow and re-dispatch them into the Grid.

7 Conclusion

We developed a scientific workflow system based on web services, SWAMP, for application users to conveniently assemble, execute, monitor, and control complex computing workflows in heterogeneous network environments. We proposed two efficient mapping algorithms, impRCP for MED and Greedy LDP for MFR, and incorporated them into SWAMP for performance optimization. The system was implemented, deployed, and tested for a real-life scientific application in Open Science Grid.

We learned a valuable lesson in the real system implementation that the workflow performance highly relies on the quality of the mapping scheme, which in turn relies on the accuracy of the performance modeling and estimation. The time-varying nature of system and network resources' availability makes it very challenging to perform an accurate estimation of the execution time of a module or the transfer time over a link in real networks. We will investigate sophisticated cost models to characterize real-time node and link behaviors in dynamic network environments. Also, the overhead brought by Stork may counteract the performance improvement of distributed execution in small problem sizes, but it is worth the effort for workflows with large data sizes, which are common in large-scale scientific applications.

Acknowledgements This research is sponsored by U.S. Department of Energy's Office of Science under Grant No. DE-SC0002400 with University of Memphis and Grant No. DE-SC0002078 with Southern Illinois University at Carbondale. Lin and Liu are supported by the DOE Earth System Modeling (ESM) Program via the FASTER project. We would like to thank two anonymous reviewers for their insightful and constructive comments.

References

1. Afrati, F., Papadimitriou, C., Papageorgiou, G.: Scheduling DAGs to minimize time and communication. In: Proc. of the 3rd Aegean Workshop on

Computing: VLSI Algorithms and Architectures, pp. 134–138. Springer, Berlin (1988)
2. Agarwalla, B., Ahmed, N., Hilley, D., Ramachandran, U.: Streamline: a scheduling heuristic for streaming application on the Grid. In: Proc. of the 13th Multimedia Comp. and Net. Conf. San Jose, CA (2006)
3. Ahmed, I., Kwok, Y.: On exploiting task duplication in parallel program scheduling. IEEE Trans. Parallel Distrib. Syst. **9**, 872–892 (1998)
4. Annie, S., Yu, H., Jin, S., Lin, K.-C.: An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Trans. Parallel Distrib. Syst. **15**, 824–834 (2004)
5. Bandwidth Test Controller: http://www.internet2.edu/performance/bwctl/. Accessed 1 Aug 2012
6. Boeres, C., Filho, J., Rebello, V.: A cluster-based strategy for scheduling task on heterogeneous processors. In: Proc. of 16th Symp. on Comp. Arch. and HPC, pp. 214–221 (2004)
7. Bozdag, D., Catalyurek, U., Ozguner, F.: A task duplication based bottom-up scheduling algorithm for heterogeneous environments. In: Proc. of the 20th IPDPS (2006)
8. Benoit, A., Robert, Y.: Mapping pipeline skeletons onto heterogeneous platforms. JPDC **68**(6), 790–808 (2008)
9. Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor, I., Wang, I.: Programming scientific and distributed workflow with triana services. Concurrency and Computation: Practice and Experience, Special Issue: Workflow in Grid Systems **18**(10), 1021–1037 (2006). http://www.trianacode.org
10. Climate and Carbon Research Institute: http://www.ccs.ornl.gov/CCR. Accessed 1 Aug 2012
11. Cordella, L., Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. In: Proc. of the 3rd Int. Workshop on Graph-based Representations, Italy (2001)
12. DAGMan: http://www.cs.wisc.edu/condor/dagman. Accessed 1 Aug 2012
13. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proc. of 6th Symp. on Operating System Design and Implementation, San Francisco, CA (2004)
14. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-Science: an overview of workflow system features and capabilities. J. of Future Generation Comp. Sys. **25**(5), 528–540 (2009)
15. Deelman, E., Singh, G., Su, M., Blythe, J., Gil, A., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C., Katz, D.S.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci. Program. **13**, 219–237 (2005)
16. Dhodhi, M., Ahmad, I., Yatama, A.: An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems. JPDC **62**, 1338–1361 (2002)
17. Distributed computing projects: http://en.wikipedia.org/wiki/List_of_distributed_computing_projects. Accessed 1 Aug 2012
18. Dobber, M., van der Mei, R., Koole, G.: A prediction method for job runtimes on shared processors: survey, statistical analysis and new avenues. Perform. Eval. **64**(7–8), 755–781 (2007)
19. Earth Simulator Center: http://www.jamstec.go.jp/esc. Accessed 1 Aug 2012
20. Earth System Grid (ESG): http://www.earthsystemgrid.org. Accessed 1 Aug 2012
21. Fahringer, T., Prodan, R., Duan, R., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.-L., Villazon, A., Wieczorek, M.: ASKALON: a Grid application development and computing environment. In: Proc. of the 6th IEEE/ACM Int. Workshop on Grid Comp., pp. 122–131 (2005)
22. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
23. Gates, M., Warshavsky, A.: Iperf version 2.0.3. http://iperf.sourceforge.net. Accessed 1 Aug 2012
24. Gerasoulis, A., Yang, T.: A comparison of clustering heuristics for scheduling DAGs on multiprocessors. JPDC **16**(4), 276–291 (1992)
25. Globus Replica Location Service: http://www.globus.org/toolkit/data/rls/. Accessed 1 Aug 2012
26. GridFTP: http://www.globus.org/grid_software/data/gridftp.php. Accessed 1 Aug 2012
27. Gu, Y., Wu, Q.: Maximizing workflow throughput for streaming applications in distributed environments. In: Proc. of the 19th Int. Conf. on Comp. Comm. and Net., Zurich, Switzerland (2010)
28. Gu, Y., Wu, Q.: Optimizing distributed computing workflows in heterogeneous network environments. In: Proc. of the 11th Int. Conf. on Distributed Computing and Networking, Kolkata, India, 3–6 Jan 2010
29. Gu, Y., Wu, Q., Benoit, A., Robert, Y.: Optimizing end-to-end performance of distributed applications with linear computing pipelines. In: Proc. of the 15th Int. Conf. on Para. and Dist. Sys., Shenzhen, China, 8–11 Dec 2009
30. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. Nucleic Acids Res **34**, 729–732 (2006). http://www.taverna.org.uk
31. Ilavarasan, E., Thambidurai, P.: Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. J. Comp. Sci. **3**(2), 94–103 (2007)
32. Johnston, W.: Computational and data Grids in large-scale science and engineering. J. of Future Generation Comp. Sys. **18**(8), 1085–1100 (2002)
33. Kacsuk, P., Farkas, Z., Sipos, G., Toth, A., Hermann, G.: Workflow-level parameter study management in multi-Grid environments by the P-GRADE Grid portal. In: Int. Workshop on Grid Computing Enviornments (2006)
34. Kwok, Y., Ahmad, I.: Dynamic critical-path scheduling: An effective technique for allocating task graph to multiprocessors. IEEE Trans. Parallel Distrib. Syst. **7**(5), 506–521 (1996)

35. Kwok, Y., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. **31**(4), 406–471 (1999)
36. Large Hadron Collider (LHC): http://lhc.web.cern.ch/lhc. Accessed 1 Aug 2012
37. Laszewski, G., Hategan, M.: Workflow concepts of the Java CoG Kit. J. Grid Computing **3**(3–4), 239–258 (2005)
38. Lewis, T., EI-Rewini, H.: Introduction to Parallel Computing. Prentice Hall, New York (1992)
39. Load Sharing Facility: http://www.platform.com/workload-management/high-performance-computing/lp. Accessed 1 Aug 2012
40. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience **18**(10), 1039–1605 (2006)
41. Ma, T., Buyya, R.: Critical-path and priority based algorithms for scheduling workflows with parameter sweep tasks on global Grids. In: Proc. of the 17th Int. Symp. on Computer Architecture on HPC, pp. 251–258 (2005)
42. McCreary, C., Khan, A., Thompson, J., McArdle, M.: A comparison of heuristics for scheduling DAGs on multiprocessors. In: Proc. of the 8th ISPP, pp. 446–451 (1994)
43. McDermott, W., Maluf, D., Gawdiak, Y., Tran, P.: Airport simulations using distributed computational resources. J. Defense Soft. Eng. **16**(6), 7–11 (2003)
44. Messmer, B.: Efficient graph matching algorithms for preprocessed model graphs. PhD thesis, Institute of Computer Science and Applied Mathematics, University of Bern (1996)
45. Monitoring and Discovery System (MDS): http://www.globus.org/toolkit/mds/. Accessed 1 Aug 2012
46. Network weather service: http://nws.cs.ucsb.edu. Accessed 1 Aug 2012
47. One-Way Active Measurement Protocol: http://www.internet2.edu/performance/owamp/. Accessed 1 Aug 2012
48. Open Science Grid: http://www.opensciencegrid.org. Accessed 1 Aug 2012
49. OSCARS: On-demand Secure Circuits and Advance Reservation System: http://www.es.net/oscars. Accessed 1 Aug 2012
50. OSG Resource and Site Validation: http://vdt.cs.wisc.edu/components/osg-rsv.html. Accessed 1 Aug 2012
51. Performance Inspector: http://perfinsp.sourceforge.net. Accessed 1 Aug 2012
52. perfSONAR: http://www.perfsonar.net/. Accessed 1 Aug 2012
53. Portable Batch System: http://www.pbsworks.com/. Accessed 1 Aug 2012
54. Rahman, M., Venugopal, S., Buyya, R.: A dynamic critical path algorithm for scheduling scientific workflow applications on global Grids. In: Proc. of the 3rd IEEE Int. Conf. on e-Sci. and Grid Comp., pp. 35–42 (2007)
55. Ranaweera, A., Agrawal, D.: A task duplication based algorithm for heterogeneous systems. In: Proc. of IPDPS, pp. 445–450 (2000)
56. Rao, N.S.V.: Vector space methods for sensor fusion problems. Opt. Eng. **37**(2), 499–504 (1998)
57. Reliable File Transfer: http://www-unix.globus.org/toolkit/docs/3.2/rft/index.html. Accessed 1 Aug 2012
58. Sekhar, A., Manoj, B., Murthy, C.: A state-space search approach for optimizing reliability and cost of execution in distributed sensor networks. In: Proc. of Int. Workshop on Dist. Comp., pp. 63–74 (2005)
59. Shroff, P., Watson, D., Flann, N., Freund, R.: Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. In: Proc. of Heter. Comp. Workshop, pp. 98–104 (1996)
60. Singh, M., Vouk, M.: Scientific workflows: scientific computing meets transactional workflows. In: Proc. of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions, pp. 28–34. Univ. Georgia, Athens, GA (1996)
61. Spallation Neutron Source: http://neutrons.ornl.gov, http://www.sns.gov. Accessed 1 Aug 2012
62. Storage Resource Broker (SRB): http://www.sdsc.edu/srb/index.php/Main_Page. Accessed 1 Aug 2012
63. Storage Resource Management (SRM): https://sdm.lbl.gov/srm-wg/. Accessed 1 Aug 2012
64. Stork: http://www.cct.lsu.edu/~kosar/stork/index.php. Accessed 1 Aug 2012
65. Swift: http://www.ci.uchicago.edu/swift/main/. Accessed 1 Aug 2012
66. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M. (eds.): Workflows for e-Science: Scientific Workflows for Grids. Springer, Berlin Heidelberg New York (2007)
67. TeraPaths: https://www.racf.bnl.gov/terapaths. Accessed 1 Aug 2012
68. The Whetstone Benchmark: http://www.roylongbottom.org.uk/whetstone.htm. Accessed 1 Aug 2012
69. Topcuoglu, H., Hariri, S., Wu, M.: Performance effective and low-complexity task scheduling for heterogeneous computing. IEEE TPDS **13**(3), 260–274 (2002)
70. Wang, L., Siege, H., Roychowdhury, V., Maciejewski, A.: Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. JPDC **47**, 8–22 (1997)
71. Wassermann, B., Emmerich, W., Butchart, B., Cameron, N., Chen, L., Patel, J.: Workflows for e-Science: Scientific Workflows for Grids, Chapter Sedna: A BPEL-based Environment for Visual Scientific Workflow Modeling, pp. 427–448. Springer, London (2007)
72. Worldwide LHC Computing Grid (WLCG): http://lcg.web.cern.ch/LCG. Accessed 1 Aug 2012
73. Wu, Q., Gu, Y.: Optimizing end-to-end performance of data-intensive computing pipelines in heterogeneous network environments. J. Parallel Distrib. Comput. **71**(2), 254–265 (2011)
74. Wu, Q., Gu, Y., Liao, Y., Lu, X., Lin, Y., Rao, N.: Latency modeling and minimization for large-scale

scientific workflows in distributed network environments. In: The 44th Annual Simulation Symposium (ANSS11), Part of the 2011 Spring Simulation Multiconference (SpringSim11), Boston, MA, 4–7 Apr 2011
75. Wu, Q., Rao, N.S.V.: On transport daemons for small collaborative applications over wide-area networks. In: Proc. of the 24th IEEE Int. Performance Computing and Communications Conf., pp. 159–166, Phoenix, AZ, 7–9 Apr 2005
76. Wu, Q., Zhu, M., Lu, X., Brown, P., Lin, Y., Gu, Y., Cao, F., Reuter, M.: Automation and management of scientific workflows in distributed network environments. In: Proc. of the 6th Int. Workshop on Sys. Man. Tech., Proc., and Serv., Atlanta, GA, 19 Apr 2010